Représentation de la Connaissance
Répétition 4
17 octobre 2006

Astuce du jour

Les versions récentes de Prolog n'affichent plus qu'une partie des très longues listes. Exécuter le prédicat

```
set_prolog_flag(toplevel_print_options,
  [quoted(true), portray(true), max_depth(0)]).
```

permet de désactiver ce gadget ennuyeux.

Exercices sur les arbres

Correction des exercices proposés

1. `parcours`
Définir un prédicat `parcours(Tr,Ls)`, où `Tr` est un arbre binaire complet dont les feuilles sont étiquetées par des naturels et `Ls` est une liste, vrai si et seulement si `Ls` est la liste des feuilles rencontrées lors d'un parcours "droite - gauche" de l'arbre, mais en remplacant les feuilles étiquetées d'un nombre impair par le nombre pair directement supérieur. On demande que, dans le cas où `Ls` n'est pas complètement instancié mais que `Tr` l'est, l'instance de `Ls` telle que `parcours(Tr,Ls)` est vrai soit générée. On représentera les arbres de la même manière que dans l'exercice précédent.

Exemple(s) d'utilisation:

```
parcours([1|[[2|[3|4]]|[5|6]]],Y). => N = [6,6,4,4,2,2]
```

2. `depth_list`
Définir un prédicat `depth_list(Lss,N)`, où `Lss` est une liste dont les éléments sont des atomes ou des listes dont les éléments sont des atomes ou des listes dont ... et `N` est un naturel, vrai si et seulement si la liste `Lss` a une profondeur égale à `N`. On demande que, dans le cas où `N` n'est pas complètement instancié mais que `Lss` l'est, l'instance de `N` telle que `depth_list(Lss,N)` est vrai soit générée.

Exemple(s) d'utilisation:

```
depth_list([[[1],2,[3,[4]]],[]],N). => N = 4
depth_list([],N). => N = 0
```

3. `same_frame`

Définir le prédicat `same_frame(Tr1,Tr2)` vrai si et seulement si les arbres binaires complets `Tr1` et `Tr2` ont le même ensemble de feuilles (dans un premier temps, avec le même nombre d'occurrences et, dans un second temps, sans cette contrainte).

4. `simplify`

Définir le prédicat `simplify(Tr1,Tr2)` vrai si et seulement si `Tr1` et `Tr2` sont des arbres binaires complets dont les feuilles sont étiquettées par des naturels et que `Tr2` est la version simplifiée de `Tr1` obtenue en remplacant les sous-arbres ayant deux feuilles étiquettées par le même naturel par ce naturel. Dans un premier temps, on ne simplifiera qu'à un niveau de l'arbre, ensuite on simplifiera récursivement tant que les étiquettes obtenues coïncident. On demande que, dans le cas où `Tr2` n'est pas complètement instancié mais que `Tr1` l'est, l'instance de `Tr2` telle que `simplify(Tr1,Tr2)` est vrai soit générée.

Exercices sur la coupure

5. La coupure

Soit le programme suivant

```
a(X) :- b(X).                    c(1).   d(2).
a(X) :- c(X).                    e(3).   f(3).
b(X) :- d(X).                    e(4).   f(4).
b(X) :- e(X),!,f(X).             e(5).   f(5).
b(X) :- g(X).                    g(6).   h(7).
b(X) :- h(X).
```

Sans la coupure, que ce serait-il passé si on avait posé la question a(X).

Avec la coupure, quel sera le nouveau comportement ?

Dessiner l'arbre de recherche et indiquer les branches qui seront coupées par la coupure.

Effets inattendus de la coupure

6. `nombre_de_parents`
En utilisant la coupure, définir le prédicat `nombre_de_parents(X,N)` vrai si et seulement si `N` est le nombre de parents de `X`. On rappelle que tout individu a deux parents, sauf Adam et Eve, qui n'en n'ont pas, et Jésus qui n'en a qu'un. On demande que le prédicat fonctionne quand tout est instancié et quand le nom de l'individu est instancié mais pas le nombre de parents.

Exercices proposés

7. `member`
Récrire le prédicat `member(A,Ls)` en utilisant une ou plusieurs coupures pour améliorer l'efficacité. Déterminer quels sont les modes de fonctionnement de ce prédicat qui sont conservés, et quels sont ceux qui ont été altérés.

8. `occur`
Définir le prédicat `occur(Tr,Ls)` prenant comme arguments une paire pointée `Tr` considérée comme un arbre binaire complet et une liste de paires pointées `Ls`, vrai si et seulement si chaque paire pointée de `Ls` a comme membre de gauche une étiquette d'une feuille de `Tr` et comme membre de droite le nombre d'occurences de cette étiquette dans l'arbre. Chaque étiquette de `Tr` n'apparaît qu'une seule fois dans `Ls` et ce dans l'ordre d'apparition gauche-droite des feuilles dans l'arbre.

Exercices arithmétiques

Correction des exercices proposés

9. `pgcd`
Définir le prédicat `pgcd` vrai si et seulement si le troisième argument est le plus grand diviseur commun des deux premiers arguments. On demande que quand le troisième argument n'est pas instancié, mais que les deux premiers le sont, le plus grand diviseur commun soit retourné.

10. `perfect`
Définir le prédicat `perfect` vrai si et seulement si l'argument est un nombre parfait. Pour rappel, un nombre est parfait si la somme de ses diviseurs (excepté lui-même) rend le nombre.

Exemples :

```
6 = 1+2+3
28 = 1+2+4+7+14
```

Dans un premier temps le prédicat ne permettra que de tester si un nombre est parfait, dans un deuxième temps on générera les nombres parfaits (dans l'ordre croissant) lorsque l'argument n'est pas instancié.

Exercice proposé

11. Problème du portefeuille

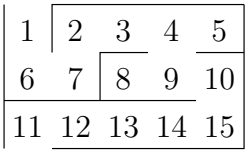
On dispose d'un ensemble de coupures (pièces d'un euro, de 2, billets de 5, 10, 20, 50, 100, 200, 500) et on demande de combien de manière différentes il est possible de payer une certaine somme.

Exercices sur les graphes

Exercice proposé

12. Labyrinthe

Considérons le labyrinthe suivant



Ecrire un programme Prolog permettant de trouver un chemin de 1 vers 11.